Real-Time Linux
for
Audio Applications

John Kacur
Red Hat

Agenda

- What is real-time
- What is real-time Linux
- Kernel Internals
- Getting and compiling the kernel
- Configuring your system
- Testing / Tuning
- Programming
- How to reach us

Real-Time Linux

- Uses
  - Controlling machines in Industry
  - Military
  - Stock Market
  - Audio and Video Applications and distributions

Real-Time Linux

- Audio and Video Distributions among the first uses of the real-time kernel for a wider audience.

Real-Time Linux

- What is real-time?
  - real-fast???
  - Deterministic
  - Predictable
  - low latency
- Determinism vs Throughput
  - “Bursty”
  - Easy to explain to musicians

Real-Time Linux

- Hard vs Soft-Real Time
  - Hard
    - mathematically provable
    - guaranteed to meet deadlines.
      - how to react if a deadline is missed?
      - deliver event anyway or not? fail? warn if a deadline won't be met?
  - Soft
    - can sometimes miss deadlines

Real-Time Linux

- What is Real-Time Linux?
  - rt patch on-top of standard kernel
  - rt-mutexes
    - preemptable
    - priority inheritance
  - high resolution timers

Real-Time Linux

- What is Real-Time Linux? (continued)
  - Threaded Soft-irqs
  - Threaded Hard-irqs
  - Preemptable RCU

Real-Time Linux

- PREEMPT_DESKTOP vs PREEMPT_RT
  - both provide involuntary preemption
- Critical Sections
  - Locks protect data from being modified by more than one processor at a time.

Real-Time Linux

- PREEMPT_DESKTOP
  - spinlocks protect critical sections on smp systems.
  - these also correspond to the sections that cannot be preempted

Real-Time Linux

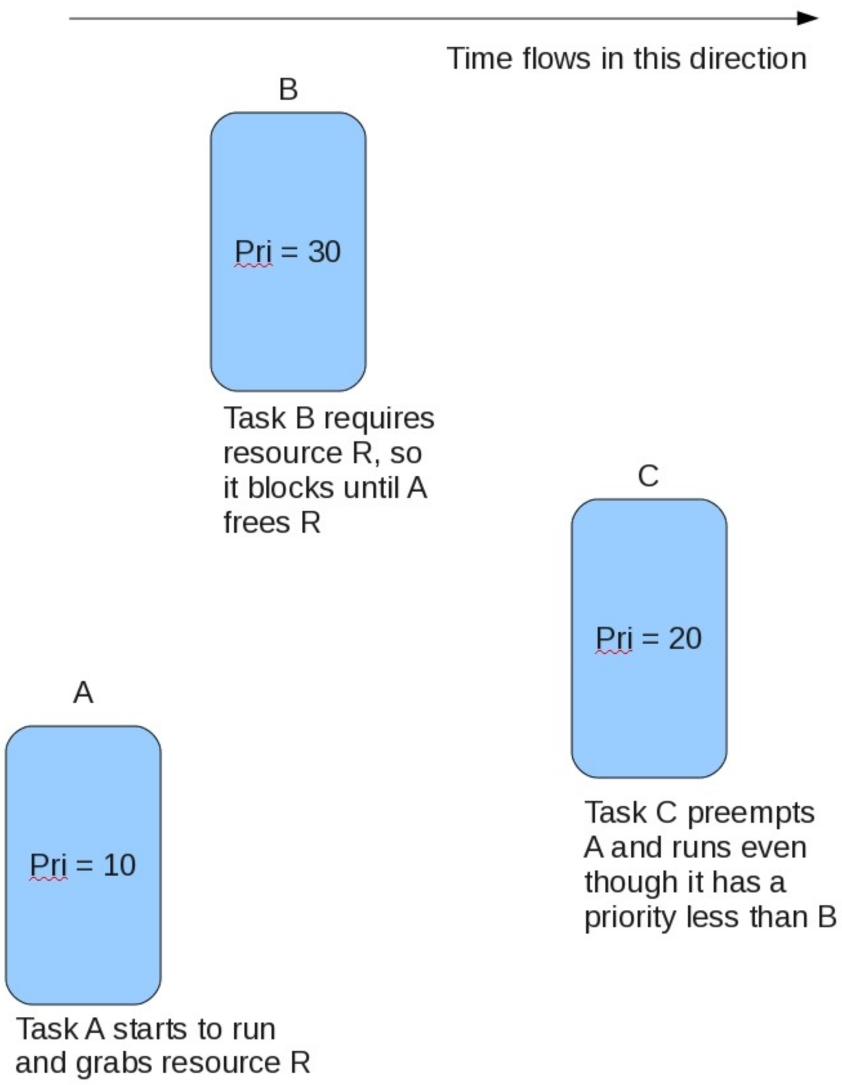
- PREEMPT_RT
  - spinlocks are converted to sleeping spinlocks, called rt mutexes
  - these critical sections are now preemptible. Threads attempting to access locked sections are scheduled out.
  - raw_spinlock_t protects sections that cannot be preempted even under PREEMPT_RT
    - arch_spinlock_t – hardware level implementation.

Real-Time Linux

- SCHED_FIFO
- SCHED_RR
  - round robin, interrupted by a time quantum
  - best for testing
- SCHED_OTHER
  - standard scheduling algorithm

Real-Time Linux

- Priority Inversion
- Priority Inheritance
  - Bounded Priority Inversion



Time flows in this direction

B

Pri = 30

Task B requires resource R, so it blocks until A frees R

C

Pri = 20

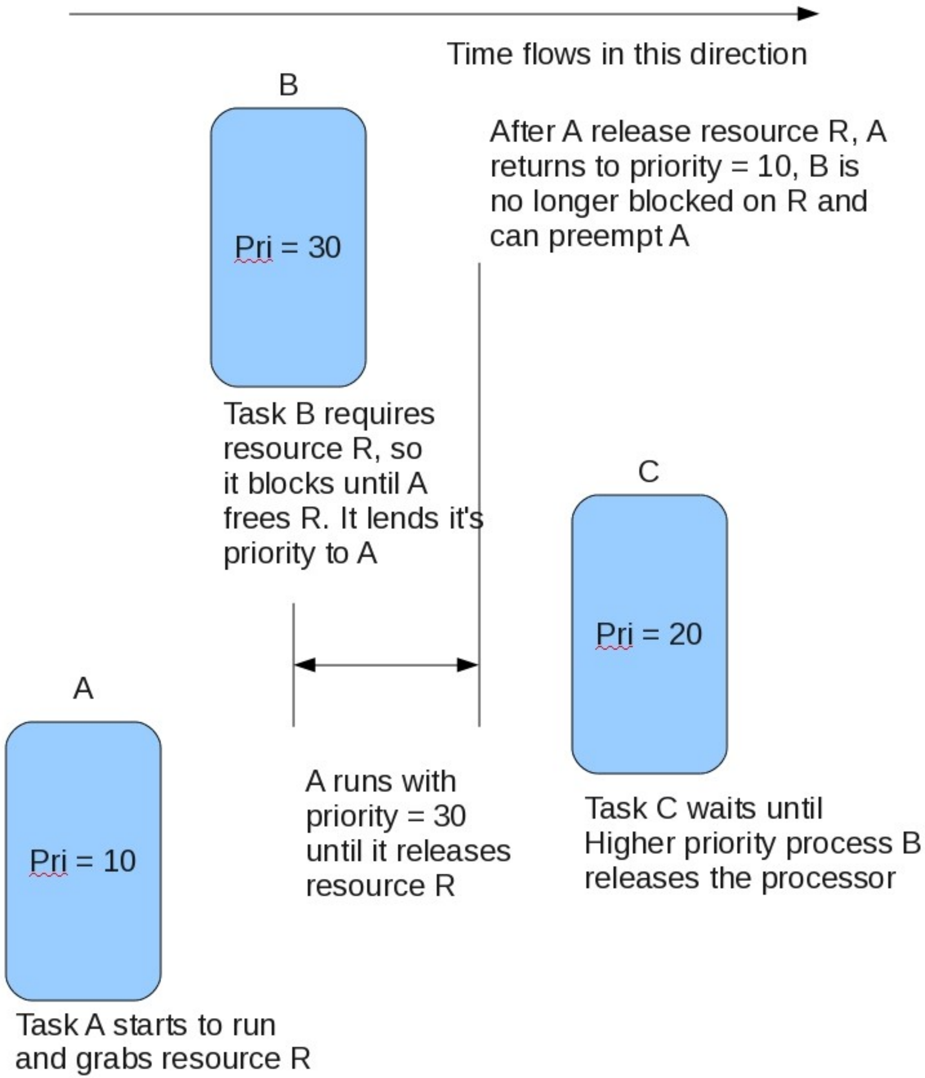
Task C preempts A and runs even though it has a priority less than B

A

Pri = 10

Task A starts to run and grabs resource R

Fig.1. Without priority inversion, C preempts A and runs at the expensive of higher priority B



Time flows in this direction

A

Pri = 10

Task A starts to run
and grabs resource R

B

Pri = 30

Task B requires
resource R, so
it blocks until A
frees R. It lends it's
priority to A

A runs with
priority = 30
until it releases
resource R

After A release resource R, A
returns to priority = 10, B is
no longer blocked on R and
can preempt A

C

Pri = 20

Task C waits until
Higher priority process B
releases the processor

Fig.2. With priority inversion, B lends it's priority to A
until A frees resource R, allowing B to run

Real-Time Linux - BKL

- BKL – rtmutex in preempt-rt
  - kind of global spin-lock with some unusual properties.
    - can be released and re obtained automatically
  - goal is to remove the bkl completely from the standard kernel.
  - requirement to get the full preempt-rt patch into the mainstream kernel.

Real-Time Linux

- Fetching and Building
  - http://www.kernel.org/pub/linux/kernel/projects/rt/
  - Eg. patch-2.6.33.2-rt13
    - kernel 2.6.33 + stable patch 2.6.33.2
  - Can use “ketchup”
    - http://people.redhat.com/srostedt/rt/tools
  - http://people.redhat.com/srostedt/rt/tools/ketchup-0.9.8-rt3

Real-Time Linux

- tar xjf linux-2.6.33.tar.bz
- cd linux-2.6.33
- bunzip2 -c patch-2.6.33.2 | patch -p1
- bunzip2 -c patch-2.6.33.2-rt13 | patch -p1

Real-Time Linux

- Copy over your distro's config file
  - /proc/config.gz
- make silentoldconfig
- make localmodconfig
  - removes modules that are unloaded

Real-Time Linux

- Processor type and features
  - complete preemption (PREEMPT_RT)
    - Thread Softirqs (PREEMPT_SOFTIRQS)
    - Thread Hardirqs (PREEMPT_HARDIRQS)
  - SLAB, SLUB not supported
  - High Resolution Timer Support (HIGH_RES_TIMERS)

Real Time Linux

- FUNCTION_TRACER
  - PREEMPT_TRACER
  - IRQSOFF_TRACER
  - branch profiler can all add latency

Real-Time Linux

- Who gets real-time privileges?
- sudo groupadd realtime
- sudo useradd -G realtime jkacur

Real Time Linux

- /etc/security/limits.conf

```
@realtime     soft    cpu           unlimited
@realtime     -       rtprio        100
@realtime     -       nice          -20
@realtime     -       memlock           unlimited
```

Real-Time Linux - irqs

- Now that irqs are threads we can assign them priorities
-
  ```
  ps -eLo rtprio,pid,cls,rtprio,pri,nice,cmd | sort -rn
  ```
- You can even assign them cpumasks to limit them to certain processors. (in most cases)

Real-Time Linux

- /etc/rtgroups

```
kthreads:*:1:*:\[.*\]$
watchdog:f:99:*:\[watchdog.*\]
migration:f:99:*:\[migration\/.*\]
softirq:f:70:*:\[.*(softirq|sirq).*\]
softirq-net-tx:f:75:*:\[(softirq|sirq)-net-tx.*\]
softirq-net-rx:f:75:*:\[(softirq|sirq)-net-rx.*\]
softirq-sched:f:1:*:\[(softirq|sirq)-sched\/.*\]
rpciod:f:65:*:\[rpciod.*\]
lockd:f:65:*:\[lockd.*\]
nfsd:f:65:*:\[nfsd.*\]
hardirq:f:85:*:\[(irq|IRQ)[\-_/].*\]
```

Real-Time Linux

- /proc/sys/kernel/sched_rt_runtime_us
  - 950000
- /proc/sys/kernel/sched_rt_period_us
  - 1000000
- 100000 – 95000 = 50000 us or 0.05 seconds for SCHED_OTHER
  - safety
  - Can disable by echoing -1 in sched_rt_runtime_us

Real-Time Linux

- irqbalance - recommend to turn off.
  - service irqbalance stop
  - chkconfig irqbalance off
  - instead configure irq threads
- cpuspeed – recommend to turn off.
  - service cpuspeed stop
  - chkconfig cpuspeed stop
  - Caution – reduces power savings.

Real-Time Linux - rt-tests

- git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/ rt-tests.git
- cyclictest, hwlatdetect, pip_stress, pi_stress

  pmqtest, ptsematestrt-migrate-test, sendme

  signaltest, sigwaittest, svsematest, svsematest

Real-Time Linux - rt-tests

- cyclictest
  - -t numthreads -p priority

```
./cyclictest -p50 -t4
policy: fifo: loadavg: 0.02 0.11 0.24 2/321 20139

T: 0 (20134) P:50 I:1000 C:  76862 Min:     10 Act:  116 Avg:   77 Max:     197
T: 1 (20135) P:49 I:1500 C:  51242 Min:     10 Act:   34 Avg:   75 Max:     191
T: 2 (20136) P:48 I:2000 C:  38431 Min:      7 Act:   12 Avg:   68 Max:     352
T: 3 (20137) P:47 I:2500 C:  30745 Min:      9 Act:   44 Avg:   77 Max:     183
```

Real-Time Linux – rt-tests

- SMIs
  - system management interrupts
  - can be necessary, ie protect processor from overheating.
  - no control over them, sometimes written poorly.
- sudo ./hwlatdetect --duration=2m –threshold=500
  - hwlat_detect.ko
  - monopolizes a cpu with interrupts disabled

Real-Time Linux - rteval

- git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rteval.git
- cyclictest + standard benchmarks to stress your machine
  - hackbench, dbench

Real-Time Linux

- hwlatdetect
  - hwlat_detect.ko
  - disables interrupts
  - monopolizes cpu
  - stop cpu
  - sudo ./hwlatdetect --threshold=300 --duration=1m

Real-Time Linux - programming

- Simple -rt program, set a policy and priority

```
#define POLICY SCHED_FIFO

        struct sched_param param;

        print_prio_policy(0);  /* Print current priority and policy */

        /* Set a real-time policy and priority */
        param.sched_priority = sched_get_priority_min(POLICY);
        sched_setscheduler(0, POLICY, &param);

        print_prio_policy(0);  /* Print current priority and policy */
```

Real-Time Linux - programming

- Helper Functions

```
/* get the priority and policy of a given pid */
void get_prio_policy(pid_t pid, int *prio, int *policy)
{
        struct sched_param param;
        sched_getparam(pid, &param);
        *prio = param.sched_priority;
        *policy = sched_getscheduler(pid);
}

void print_prio_policy(pid_t pid)
{
        int prio, policy;
        get_prio_policy(pid, &prio, &policy);
        print_rt(pid, prio, policy);
}
```

Real-Time Linux - programming

```
./hello_rt
pid=18455, prio=0, policy=SCHED_OTHER
pid=18455, prio=1, policy=SCHED_FIFO
```

Real-Time Linux - programming

- pthreads usually have parallel apis

```
/* Common struct */
struct sched_param {
      int sched_priority;     /* Scheduling priority */
};
```

```
int sched_setscheduler(pid_t pid, int policy,
        const struct sched_param *param);
```

```
pthread_setschedparam(pthread_t thread, int policy,
        const struct sched_param *param);
```

Real-Time Linux - programming

- Locking with older UNIX apis are usually done with semaphores
  - no support for priority inheritance
- pthread locks are done with pthread mutexes
  - implemented with futexes, which are backed in the kernel by rtmutexes
  - rtmutexes provide the support for priority inheritance

Real-Time Linux - programming

- Trick for using older apis without pthreads
  - mmap a pthread_mutex in shared memory.
  - call the pthread_mutex init and locking apis on that to use for your locking.
- See pip_stress.c in rt-tests for an example of how this works.

Real-Time Linux - programming

- Swapping to disk will obviously create large latencies. Use mlock to lock down memory.
  - mlockall(MCL_CURRENT | MCL_FUTURE)
  - a little naive when program size grows though.

Real-Time Linux

- Where we hang out.
- https://rt.wiki.kernel.org/index.php/Main_Page
- IRC
  - OFTC (irc.oftc.net) #linux-rt
- mailing list
  - linux-rt-users@vger.kernel.org

Real-Time Linux

- Contacting me.
- jkacur@redhat.com
- I am often on the #linux-rt irc channel as well.
- Special thanks to Steven Rostedt for answering all of my questions.